# Software Technologies for Embedded Systems: An Industry Inventory

Bas Graaf, Marco Lormans, and Hans Toetenel

Faculty of Information Technology and Systems, Delft University of Technology
The Netherlands
{b.s.graaf,m.lormans,w.j.toetenel}@its.tudelft.nl

**Abstract.** This paper addresses the ongoing inventory activities within the ITEA MOOSE project. The inventory result will be a complete view on the application of methods, techniques and tools for software production within some of the leading European industrial companies within the embedded system field, such as Philips, Océ, ASML and Nokia. The current results are remarkable, as they confirm the cautiousness of industry to adopt recent state of the art development technologies, even as the production of in-time reliable software products becomes more and more an unreachable target.

## 1 Introduction

Embedded systems are getting more and more complex. At the same time an increasingly bigger part of embedded systems is implemented through software. This results in big challenges for developing embedded software. As developing embedded software is fundamentally different from developing non-embedded software research specifically targeted at the embedded domain is required. With the ever-increasing penetration of embedded systems in society and the related increase in investments by industry to develop such systems, the investments in embedded software engineering technologies (methods, tools, techniques, processes) increase as well.

This paper presents some results of the MOOSE (software engineering MethOdOlogieS for Embedded systems) project [1]. MOOSE is an ITEA project [2] aimed at improving software quality and development productivity in the embedded systems domain. One of the goals of this project is to integrate systems and software engineering, requirements engineering, product architecture design and analysis, software development and testing, product quality and software process improvement methodologies into *one common framework and supporting tools for the embedded domain*.

In order to create a framework of embedded software development technologies, more insight is needed in currently available methods, tools, and techniques. The main focus of this paper is the embedded software development technologies used in industry or, more precisely, used in the industrial partners of the MOOSE-consortium.

M. Oivo and S. Komi-Sirviö (Eds.): PROFES 2002, LNCS 2559, pp. 453-465, 2002.

The results of this inventory will be applied to classification schemes for embedded software product and projects that are to be developed. Finally these classification schemes must enable us to design the framework for embedded software development technologies. The framework can be seen as a structure of which methods, tools and techniques are the components. This industrial inventory cannot only contribute to the design of the framework but also can be used to add information to the framework.

Besides its use for creating the framework, the results of the inventory can also be used to determine the direction for additional research within the MOOSE project. For instance when it turns out that some technologies are missing or not completely applicable for development of embedded systems and software.

In this paper we will present the results of the inventory. The rest of this paper is organized as follows. Section two provides background on the essence of embedded systems. We define how we understand embedded systems and give a short survey of software methods, techniques and tools of today. Section three presents the main part of this paper. It describes the inventory process and the resulting product. Section four relates the work in MOOSE to other similar projects. Section five concludes the paper and presents future work.

## 2 Embedded Systems

### 2.1 What Are Embedded Systems?

As the MOOSE project is about methods, techniques and tools in the embedded systems domain, we need a definition of embedded system together with some characteristics of embedded systems. We can then decide whether a certain method, technique or tool is relevant for our project or not. Also we can use the definition to define embedded *software* as the software in an embedded system.

There is no general consensus about what an embedded system is nor is there a complete list of characteristic properties of such systems. What is generally agreed on is that an embedded system is a mixed hardware / software system dedicated to a specific application [3–7].

An embedded system is in general part of a larger system, i.e. it is a subsystem of another system. Mostly the relation with that supersystem is that the embedded system reacts on it. An embedded system is thus mostly a reactive system. This means that a car by itself is not an embedded system, nor is a mobile phone. However some subsystems of these systems possibly are (e.g. fuel injection system).

The supersystem has to be a system of a certain type before we can speak of an embedded system. For example a computer system that monitors stock rates can also be part of a larger system of a bank's trading department that involves procedures, people and stocks. We will not consider this to be an embedded system. We will only take into account systems that are embedded in other systems that are physical entities. Embedded means at least logically connected and maybe physically.

These are the most fundamental characteristics of embedded systems. Therefore we will use the following definition in this paper:

**Definition 1.** An embedded system is a mixed hardware / software system dedicated for a specific application and is part of and reactive to a larger, physical system to which it is at least logically connected.
Properties of embedded software can mostly be deduced from these characteristics. Besides that the product type also implies some specific characteristics for embedded software.

For example: controlling real world entities often implies that embedded software has real-time constraints. Also controlling real-world, physical entities means that physical damage can occur due to failure of such software. Other properties that are common and can be deduced from the fundamental characteristics or are specific for certain product types are:

- limited functionality
- hard to change
- safety / business critica
- limited resources (memory, power, time)
- long operation required
- mass produced
- short time-to-market

These characteristics, of which some are specific for embedded systems, make that developing software for such system differs from developing non-embedded software from an engineering point of view.

### 2.2 Embedded Software Methods, Techniques and Tools

The market for software engineering technologies is largely fragmented. There is no clear market leader, and there is no supplier present that fully supports the whole development chain of embedded products. There are different suppliers for requirements engineering technologies, different vendors for design tools, etc…. Most dominant is the sales of software tools. Tools imply to support or be supported by a method or technique. Some suppliers provide methodologies with their tools, while others support generic methodologies or techniques, such as UML (OMG's Unified Modeling Language) or MOF (OMG's Meta Object Facility).

Different suppliers are present in various areas of the software engineering domain. For example the most dominant vendors of analysis, modeling and design tools [8] are: Computer Associates, Oracle, Rational Software, Versata, and Sybase. While the most dominant vendors of Quality Tools [9] are: Mercury Interactive, Compuware, Rational, Empirix, IBM, Seque Software, Cyrano, Hewlett-Packard, McCabe & Associates, RadView Software, Computer Associates, and Telelogic. Another example are vendors of Configuration Management Tools [10]: Rational, MERANT, Computer Associates, SERENA Software, Telelogic, Microsoft, MKS/Mortice Kern Systems, StarBase, IBM, Technology Builders/TBI, and Hewlett-Packard. The above overview shows clearly how differentiated this market is. Furthermore, it shows a large dominance of US companies in this market domain: Europe is laying behind in this market. Close co-operation within Europe, providing solutions to integrate / connect existing technologies or construct additional technologies will enable more focused, effective and efficient, development of embedded systems.

**Table 1.** Dutch MOOSE partners

| | |
|---|---|
| ASML | lithography systems for semiconductor industry |
| Philips | consumer electronics |
| Océ | document processing systems |
| CMG | IT services |

The improvement management domain is still in its infancy. Main dominance has come from the US Software Engineering Institute's methods, of which the Capability Maturity Model is the most well-known. Experience has shown that such models always need to be customized and tailored to the respective company that applies it to improve its embedded software processes. New developments such as product assessments are not present in the market at all. Furthermore is there no commercial organization dominating the improvement management market. Most dominant players are the European IT consulting companies and tool vendors, but this is always different from country to country.

An other important technology for fast delivery of embedded systems comes from the COTS domain. Today's COTS market is rapidly changing, as the needs of companies increase. More and more components are appearing in the software market each day in order to fulfill the business demand. A quick look at COTS purchase market confirms that COTS sales are indeed getting bigger very fast every year. There exists a clear geographical distribution in software components sales (US and rest of the world). There is still a patent differentiation between the number of sales made in Europe and those reported in the US.

# 3 Inventory Results

## 3.1 Introduction

In this section we present the results of an inventory of (embedded) software engineering methods, tools and techniques used in industry. This inventory is performed as part of the MOOSE project. At the time of this writing it has been done at four Dutch industrial MOOSE partners.

It was carried out by doing interviews at different industrial MOOSE partners. First an inventory approach was made. It contained a list of topics and subtopics to be discussed. The primary focus was on the software development processes and the methods, techniques and tools used within these processes. This focus was roughly organized by the following subtopics: requirements, architecture design, modeling, coding, and testing. Besides the topics related to the software product, other items were added to get a general understanding of the type of products built. Finally there were some items about management aspects of embedded software development including items concerning project metrics and project organization.

A total of 16 respondents were interviewed at four Dutch industrial MOOSE partners. The results were processed individually, which yielded sixteen interview reports. All of the respondents were invited to provide feedback, which was processed afterwards in the results. Finally an inventory report was made per company (four

reports) on which this section is based. The four partners and the products involved are mentioned in Table 1.

The embedded software products that are built at the participating companies are very diverse. The software products were embedded in systems ranging from consumer electronics to highly specialized industrial machines.

### 3.2 PROFES and BOOTSTRAP

To make it possible to place specific techniques, methods and tools in perspective we used a software process model based on the PROFES methodology [11]. This model is suited for characterizing the development processes of embedded systems and is based on the BOOTSTRAP methodology [12, 13] for software process assessment and improvement. However, in our approach it is only used to characterize the various applied technologies.

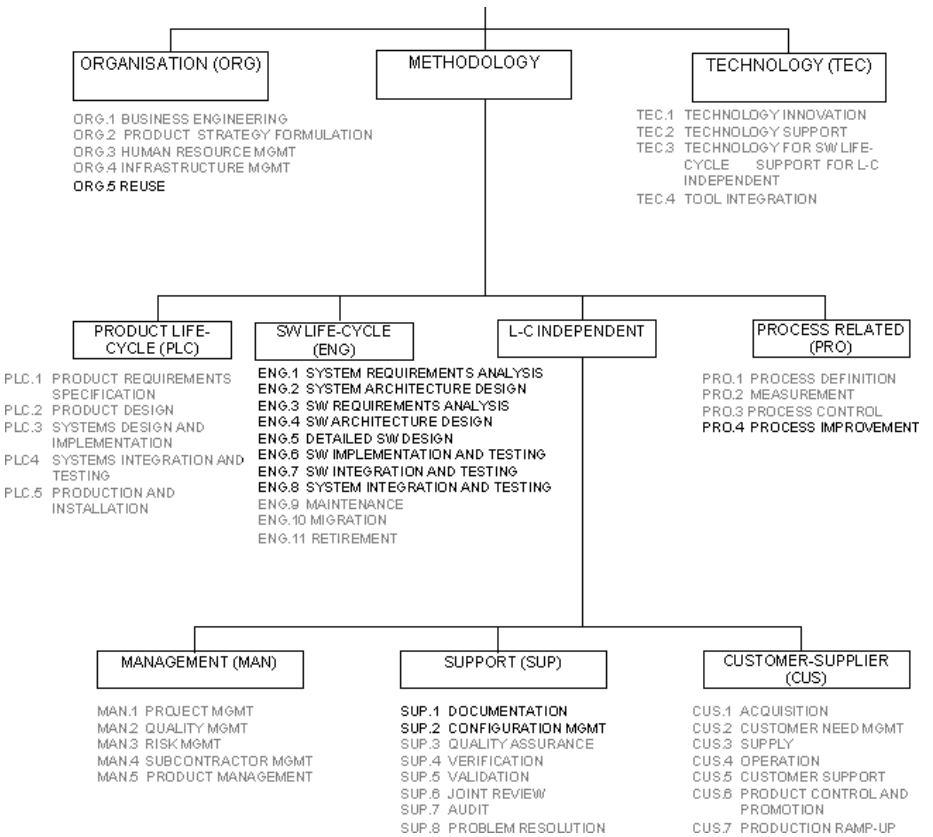


**Fig. 1.** BOOTSTRAP / PROFES processes

BOOTSTRAP is a methodology for software process assessment and improvement focused at the European industry. The BOOTSTRAP Consortium developed it during

the years 1990-93. It is a methodology comparable with SPICE [14], CMM [15, 16] and ISO 9000 [17]. Compared with the other methodologies mentioned earlier, which are more management or organization oriented [18, 19], the software process model used in BOOTSTRAP is more oriented towards software engineering processes. This makes the BOOTSTRAP process model more suited for processing the results of the inventory, which is primary focused on the software engineering technologies.

The PROFES improvement methodology was developed in the PROFES Esprit project. During this project the BOOTSTRAP process model was enhanced to better suit the embedded systems domain. As some of the MOOSE project members also participated in the PROFES project, the usage of the PROFES process model for the inventory results seemed most appropriate. The PROFES / BOOTSTRAP processes are depicted in Fig. 1.

### 3.3 Results

The results are presented here organized by the PROFES / BOOTSTRAP software development processes. Not all processes are covered because the scope of the interviews was mostly limited to the software development life-cycle. The processes that are covered bellow are printed bold in Fig. 1.

**Reuse.** In two cases reuse was formally organized within a project or company. In one case this was done in combination with the usage of the Koala [20] component model. This component model is developed for application in consumer electronics embedded software development and specially suited for facilitating reuse. It is applied together with a propriety development method, which is suited for development of product families.

In another case there was a special project in which reusable components for a certain subsystem of the product architecture were made. These components were developed as executable models with Rational Rose RealTime.

In general reuse is done rather ad-hoc. Projects reused requirements, design documents and code from earlier, similar projects just by copying them. In particular for highly specialized products it was considered not possible to make use of configurable components from a component repository.

**Life-Cycle.** Model The embedded system development process life-cycle model used is generally the V-model. This model is especially suited for development of systems comprising many (levels of) subsystems.

The Rational Unified Process (RUP [21]) or an adapted version of it was also used in a few cases. RUP is a web-enabled tool facilitating support for a set of software engineering processes.

One respondent had made a `life-cycle toolkit' for software development. The application of this toolkit guides the user through the development processes and the produced software automatically complies with some of the IEC standards (e.g. IEC 61508 for safety related systems). Also a propriety software development method called MG-R was used which enabled large-scale, multi-site and incremental software development.

**System Requirements Analysis.** System or product engineering is an activity typically performed when developing embedded systems. However in some cases system requirements and system architecture were already known in advance. This can be the case when the hardware is developed first or when the system architecture is stable for a specific product family. A feasibility study is often done in this phase. Also prototypes are built from a technical or customer perspective. This is done to establish technical feasibility or customer requirements respectively.

Sometimes use cases and UML [22] sequence diagrams are applied to express requirements on a system level. However, the meaning of UML notations has to be agreed on this level. Typical input for system requirements comes from customers or marketing, support, manufacturing, hard- and software suppliers and other stakeholders.

**System Architecture Design.** On the system level is decided what is built in software. While there are some basic guidelines for this, these decisions are often based on implicit criteria. Depending on the complexity of the product, the architecture is composed of multi- or mono-disciplinary subsystems. The subsystems are decomposed into mono-disciplinary components. This gives a tree of requirements and design documents. In this tree a design on one level is (input for) a requirement on a lower, more detailed level.

**Software Requirements Analysis.** Pre- and post conditions are a commonly used technique in specifying software requirements. This is mostly done in natural language or a semi-formal notation (e.g. pseudo code). Use cases are also used in OO-environments for specifying software requirements. In one case concerning a safety-critical system, the formal notation Z [23] was used for specification.

Often Microsoft Word templates are used to introduce some general structure in requirements. However, there still is a lot of freedom for analysts to specify requirements. As a consequence requirements from different projects can look quite different.

Real-time constraints were sometimes expressed in a separate section in the requirements documents. However, these constraints were mostly not taken into account during design. Rate monotonic scheduling analysis [24] was tried in some cases, but techniques like these are typically not used. In some cases there is a separate real-time team in a project, which is responsible for real-time aspects. Often real-time constraints are implicit in the requirements. Only one tool used was specifically suited for developing real-time systems (Rational Rose RealTime).

Other constraints that are typical for embedded software, like power consumption and memory usage, were also mostly not explicitly addressed during requirements specification and design.

In some cases Rational RequisitePro was used for requirements management. However in general this was done rather ad hoc. Hand made tables were then used for tracking down requirements to design documents and test cases.

**Software Architecture Design.** Managing the complexity of the generated software is done by making use of layered (component) architectures with well-defined interfaces. In the case were the Koala component model was used the architecture was described with a special graphical notation. Koala also provides interface and

component definition languages based on C syntax. In most cases however UML class diagrams are used for specifying the software architecture. Mostly this was done with Rational Rose.

The hardware architecture is often mirrored in the software architecture. This makes the impact of changes in hardware easier to determine. Because the hardware is developed before the software, the software often has to deal with a suboptimal hardware architecture. This is also true when dealing with third-party software that is used. Possible defects of this software have to be taken into account during design.

**Detailed Software Design.** The Unified Modeling Language (UML) is the most commonly used notation. UML is used as well in requirements as in design (architecture) documents. Even in system requirements and design it is used. In the last case however the meaning of notations has to be agreed on. For drawing UML diagrams only two tools are frequently mentioned: Microsoft Visio and Rational Rose (RealTime).

Other notations that are used for modeling are dataflow diagrams, entity-relationship diagrams, flowcharts, Hatley-Pirbhai diagrams [25] and the Koala notation for describing component architectures.

Third-party hard- and software have an impact on the requirements of an embedded system. This means that it also has to be taken into account during design of an embedded system and its software. An example of this is the use of third-party operating systems, such as Microsoft TV or OpenTV, for the development of set-top boxes. This forms a big challenge and is currently not handled well.

**Software Implementation and Testing.** Embedded software is mostly implemented using (ANSI) C. In one case C++ was used as target language for code generation. Rational Rose RealTime was used for generating the code. This is an UML software design tool in which a few concepts are added to the UML notation. With these extra concepts and class- and state diagrams it is possible to produce executable UML models. Only some code fragments have to be provided in the state diagrams. Because code is only inserted in the model itself the model and generated code are always synchronized. Traditional objections against the use of C++, such as its complexity and dynamic memory allocation, were overcome by visual development environments and instantiation of all objects during initialization.

In other cases C++ and in general OO-languages were often considered as resulting in too slow and too big programs. However sometimes the OO-paradigm is used in the design of some drivers.

Several tools are used to enhance the quality of the produced code: QA-C (QA-systems), SNiFF++ (WindRiver) and lint.

**Software Integration and Testing.** For testing different techniques are used. Test cases are mostly created manually based on the requirements specifications on different levels of the V-model. They are performed by dedicated test engineers. Only on the component level the developers do the tests themselves.

Code coverage can be measured with special tools, such as Rational Purify and Insure (Parasoft). In some case homemade tools for monitoring threads and throughput are used. Also test programs were made to automate testing, which can be

implemented using scripting languages like Perl. Typically test programs can be reused.

Failures observed during testing can be examined with use of root cause analysis techniques.

Software stubs are often used as a replacement for the hardware or the rest of the system. In one case a complete simulation of some hardware components was built to use for testing. This can be a solution when time on a target machine is not always available.

Hand made tables were used to relate test cases to requirements. In this way it can be checked that all requirements are covered by at least one test case.

**System Integration and Testing.** On higher levels the software is more often tested on the target itself. Regression tests can be used in this phase. A subset of tests that is already executed is then re-executed to ensure that changes do not have unintended side effects. Random- or monkey testing is also used when testing on the target. In one case a device that generated random infrared signals to simulate a remote control.was used for this. This is an easy to use testing technique that can be fully automated.

After testing a test report is created with the results of the tests. This is also done on lower levels. These documents generally correspond to a requirements document on a certain level.

**Measurements.** In most cases not much is measured concerning process and product. Project duration and the number of change requests and problem reports are measured in most projects. Lines of code made and changed are measured using tools like QA-C.

**Process Improvement.** In one case a process improvement project was defined. It used the six key process areas of CMM level 2 as a blue print. Baseline measurements were taken and project leaders had to periodically report on estimated end date, effort spent and progress. It turned out that different groups of software developers (e.g. embedded vs. non-embedded) required a different approach to create commitment for such a project.

**Documentation.** Documentation is generally created using Microsoft Word. Not much automation was seen here. In one case LaTeX documentation containing requirements and design could be generated from a development environment based on SDW (BWise).

**Configuration Management.** Mostly all documents created during the development process and also other items, such as tools and platforms are subject to configuration management. However, in many cases documents are not updated properly. This leads to situations were only the code is up-to-date.

Various tools are used for configuration management: Rational Clearcase, Contineous CM Synergy (Telelogic) and propriety tooling.

For change management often an approach with change requests and problem reports is used, which can be managed through Rational ClearQuest.

## 4 Related Work

The MOOSE project complements the ongoing research IST projects PECOS, TOGETHER and DISCOMP by providing a comprehensive methodology deriving functional and quality requirements from the systems engineering to requirements engineering and architecting.

PECOS will provide a meta model for component and architecture specifications of embedded software, a component repository and interactive composition environment for assembling and testing embedded software of automation devices. Contrary to PECOS, MOOSE provides design and analysis methods for the development and validation of intermediate artifacts of embedded product lines, a set of different products that embody common functional and quality requirements and software architecture.

The TOGETHER project aims at formal specifications and higher quality of code by using code generation as a means of transferring formal CASE models of components to target embedded code. Product quality assessment and validation of the intermediate products of product lines are omitted in both cases, PECOS and TOGETHER.

Furthermore, MOOSE focuses on improvement management of products and processes of different types of embedded systems whereas DISCOMP focuses on process improvement and component based distributed design using object-orientation principles and tools for design and implementation of sensor based measurement software.

COTS evaluation and metrics of software development processes are also topics that are not covered by other research projects. Furthermore, are results from the already finalized IST projects PROFES, SCOPE and BOOTSTRAP applicable and will be used as input methodologies to some of the MOOSE work packages. Especially the PROFES project is a high potential in this, as the results are dedicated to the embedded domain, publicly available, and focused to establishing relationships between embedded product quality and the underlying software engineering processes.

## 5 Conclusions and Future Work

The preceding overview of our preliminary observations is only a high-level description of some of the methods, tools and techniques used for various software development processes. We conclude with some general remarks.

The used methods, tools and techniques do not only vary between the companies, but also within the companies themselves. Mostly there is a general high-level approach present, but not much is standardized on a more detailed level. So different projects often use different tools and notations. Many differences between companies and projects can be explained when looking at specific product characteristics. But still that is not always sufficient for explaining the differences.

Another remarkable observation was that the methods, tools and techniques used were rather common software engineering tools. We expected some more specialized tools were used in this area.

Finally real-time, power and memory constraints were far less prominent in software development as we expected. This could of course be related to our previous observation.

Comparing Sect. 2.2 and Sect. 3.3 we see that there is a relatively large gap between what is available and what is used in the companies we visited. The question: "Why?" is interesting in this context. However, it was not sufficiently answered during the interviews. Partly because this was not an objective when conducting the interviews. However, some things were mentioned about it. Sometimes techniques and tools were considered not mature enough for application in real-world situations (e.g. code-generation). Another explanation was the complexity, which made it too hard to apply specific tools or techniques (e.g. simulation, formal methods). Limited management support was also a success-factor in the case of SPI. Even sentimental reasons were suggested.

In literature some results can be found of the research on the (non-) usage of CASE (Computer-Aided Software/Systems Engineering) tools. We will assume that these results can be extended to the use of methods, techniques and tools in general.

In [26] a number of possible factors affecting usage and effectiveness of CASE are presented and their impact determined by regression analysis. One interesting result is the positive correlation between CASE usage and effectiveness. Even more interesting is the resulting self-reinforcing cycle of CASE usage: higher CASE usage implies higher CASE effectiveness, which implies higher perceived relative advantage. This higher-relative advantage has a positive correlation with CASE usage and thus the cycle is complete.

In [27] among others the following factors affecting CASE adoption were found: complexity, online help, ease of use and ease of learning.

Another aspect of this not covered in these articles is that it is hard to select the appropriate methods, tools and techniques in a specific situation. Especially when there is much available. This would clearly not have a stimulating effect on the use of modern technologies.

Assuming that the use of current methods, tools and techniques is a good practice it would be worthwhile to take away some of the preventing factors for adopting and using these technologies. A framework for embedded software development methods, tools and techniques can help in this. It could take away some of the preventing factors mentioned above.

Besides its main purpose, providing a means of selecting the appropriate methods tools and techniques in specific situations, it can also reduce the complexity these technologies. Of-course it cannot actually change the complexity of methods, tools and techniques. The perceived complexity however can be reduced by it. This by providing background information and case examples of application of the method, tool or technique involved. This is also related to ease of use and ease of learning.

Another factor mentioned in [26] is expectation realism. It is easily seen that a framework could have a good impact on expectation realism.

The inventory will be continued for MOOSE partners in Finland and Spain. Also a second series of inventory sessions will be done at the Dutch MOOSE-partners. These sessions will focus more on software process improvement, quality assurance and project management activities.

Besides that the shortcomings of the existing methods, tools and techniques will be discussed in more detail with people from the field. These discussions will focus on requirements and design. The results will be used to develop possible solutions for one or more problems encountered. Finally experiments will be defined to be carried out at one or more of the industrial partners to test the proposed solution in real-world situations.

## References

1. MOOSE homepages. http://www.mooseproject.org, 2002.
2. ITEA. http://www.itea-office.org, 2002.
3. Michael Barr. *Programming Embedded Systems in C and C++*. O'Reilly, first edition, 1999.
4. Sanjaya Kumar, James H. Aylor, Barry W. Johnson, et al. *The Codesign of Embedded Systems: A Unified Hardware/Software Representation*. Kluwer Academic Publishers, 1996.
5. Juan Carlos López, Román Hermida, and Walter Geisselhardt. *Embedded Systems Design and Test*. Kluwer Academic Publishers, 1998.
6. Jean Paul Calvez. *Embedded Real-Time Systems: A Specification and Design Methodology*. Wiley Series in Software Engineering Practice. John Wiley & Sons, 1993.
7. Pasi Kuvaja, Jari Maansaari, Veikko Seppänen, et al. Specific requirements for assessing embedded product development. In *International Conference on Product Focused Software Process Improvement*, pages 68–85. VTT Electronics and University of Oulu, Finland, June 1999. http://www.inf.vtt.fi/pdf/symposiums/1999/S195.pdf.
8. Analysis, modeling and design tools market forecast and analysis, 2001–2005. International Data Corporation, report IDC #24809.
9. The distributed automated software quality tools market forecast and analysis, 2001–2005. International Data Corporation, report IDC #25176.
10. Software configuration management tools forecast and analysis, 2001–2005. International Data Corporation, report IDC #24811
11. Profes Home Page. http://www.ele.vtt.fi/profes, 2002.
12. Pasi Kuvaja, Jouni Similä, Lech Krzanik, et al. *Software Process Improvement: The BOOTSTRAP Approach*. Blackwell Publishers, 1994.
13. BOOTSTRAP Institute. http://www.bootstrap-institute.com, 2002.
14. ISO/IEC JTC 1/SC 7/WG 10. SPICE: Software Process Improvement and Capability dEtermination Website. http://www.sqi.gu.edu.au/spice, 2002.
15. Software Engineering Institute Carnegie Mellon University. *The Capability Maturity Model: Guidelines for Improving the Software Process*. Addison-Wesley Publishing Company, 1995
16. CMMI Production Team. Capability Maturity Model Integration, version 1.1. Technical Report CMU/SEI-2002-TR-012, Carnegie Mellon University, Software Engineering Institute, March 2002.
17. Östen Oskarsson and Robert L. Glass. *An ISO9000 Approach To Building Quality Software.*Prentice Hall PTR, 1996.

18. Y. Wang, I. Court, M. Ross, et al. Quantitative evaluation of the SPICE, CMM, ISO 9000 and BOOTSTRAP. In Proceedings of the Third IEEE International Software Engineering Standards Symposium and Forum (ISSES97): Emerging International Standards, pages 57–68. IEEE Computer Society, June 1997.
19. Yingxu Wang, Graham King, Hakan Wickberg, et al. What the software industry says about the practices modelled in current software process models? In *Proceedings of the 25th EUROMICRO Conference,* volume 2, pages 162–168. IEEE Computer Press, 1999
20. Rob van Ommering, Frank van der Linden, Jeff Kramer, et al. The Koala component model for consumer electronics software. *IEEE Computer*, 33(3):78–85, March 2000.
21. Rational Software Corporation. Rational Unified Process. http://www.rational.com/products/rup, 2002.
22. OMG. OMG Unified Modeling Language Specification, version 1.4. http://www.omg.org/technology/documents/formal/uml.html, September 2001.
23. The Z notation. http://www.afm.fbu.ac.uk/z, 2002
24. C.L. Liu and James Layland. Scheduling algorithms for multiprogramming in a hard real-time environment. *Journal of the Association for Computing Machinery*, 20(1):46–61, January 1973
25. Derek J. Hatley and Imtiaz A. Pirbhai. *Strategies for Real-Time System Specification*. Dorset House Publishing, 1987.
26. Juhani Ivari. Why are CASE tools not used? *Communications of the ACM*, 39(10):94–103, October 1996
27. David Finnigan, Elizabeth A. Kemp, and Daniela Mehadjiska. Towards an ideal CASE tool. In *Proceedings of the International Conference on Software Methods and Tools (SMT2000)*, pages 189–197. IEEE, November 2000.